

# Accelerating Unstructured Mesh Applications using Custom Streaming Architectures

Kyrylo Tkachov

Imperial College London  
kt208@doc.ic.ac.uk

25 June 2012

Supervisor: Prof. Paul Kelly  
2nd marker: Dr. Tony Field

## Unstructured meshes

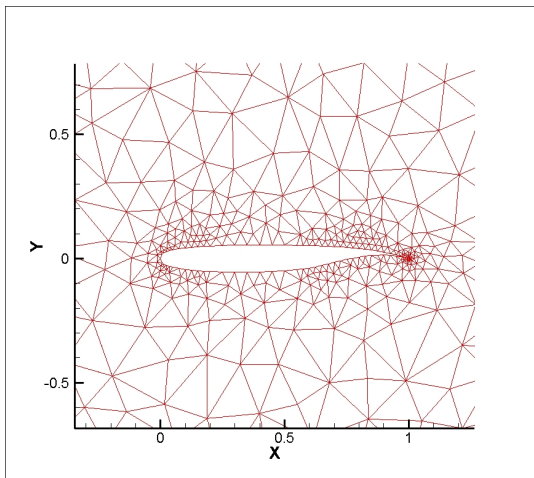


Image from Department of Environmental Engineering, University of Genoa

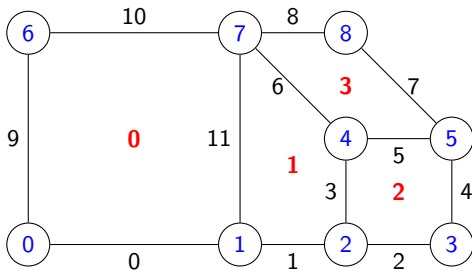
## Airfoil: Indirection maps

Elements:

- Nodes
- Cells
- Edges

edge-to-node map = {0,1, 1,2, 2,3, 2,4, 3,5, 4,5, 4,7, 5,8, 7,8, 0,6, 6,7}

cell-to-node map = {0,9,10,11, 1,2,4,7, 2,3,4,5, 4,5,7,8}



## Airfoil: Data sets

| Data set name | Associated with | Type/Dimension   |
|---------------|-----------------|--|
| x             | Nodes           | $\mathbb{R} \times \mathbb{R}$                                     |
| q             | Cells           | $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ |
| q_old         | Cells           | $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ |
| res           | Cells           | $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ |
| adt           | Cells           | $\mathbb{R}$   |
| bound         | Edges           | $\{0, 1\}$   |

## Airfoil: Kernels

| Kernel Name | Iterates over    | Reads            | Writes |
|-------------|------------------|------------------|--------|
| save_soln   | Cells            | q                | q_old  |
| adt_calc    | Cells            | x, q             | adt    |
| res_calc    | Edges            | x, q, adt        | res    |
| bres_calc   | (Boundary) Edges | x, q, adt, bound | res    |
| update      | Cells            | q_old, adt, res  | q, res |

```

void res_calc(float *x1, float *x2, float *q1, float *q2,
              float *adt1, float *adt2, float *res1, float *
              res2) {
    float dx, dy, mu, ri, p1, vol1, p2, vol2, f;
    dx = x1[0] - x2[0];
    dy = x1[1] - x2[1];
    ri = 1.0f/q1[0];
    p1 = gm1*(q1[3]-0.5f*ri*(q1[1]*q1[1]+q1[2]*q1[2]));
    vol1 = ri*(q1[1]*dy - q1[2]*dx);
    ri = 1.0f/q2[0];
    p2 = gm1*(q2[3]-0.5f*ri*(q2[1]*q2[1]+q2[2]*q2[2]));
    vol2 = ri*(q2[1]*dy - q2[2]*dx);
    mu = 0.5f*((*adt1)+(*adt2))*eps;
    f = 0.5f*(vol1* q1[0] + vol2* q2[0] ) + mu*(q1[0]-q2[0]);
    res1[0] += f;
    res2[0] -= f;
    f = 0.5f*(vol1* q1[1] + p1*dy + vol2* q2[1] + p2*dy) + mu*(q1
        [1]-q2[1]);
    res1[1] += f;
    res2[1] -= f;
    f = 0.5f*(vol1* q1[2] - p1*dx + vol2* q2[2] - p2*dx) + mu*(q1
        [2]-q2[2]);
    res1[2] += f;
    res2[2] -= f;
    f = 0.5f*(vol1*(q1[3]+p1) + vol2*(q2[3]+p2) ) + mu*(q1[3]-q2[3])
        ;
    res1[3] += f;
    res2[3] -= f;
}

```

## res\_calc data requirements

- Iterates over edges
- Processing each edge requires 2 cells, 2 nodes.
- Each edge **increments** two cells ( $+=$ ).
- Most computationally intensive kernel in Airfoil.

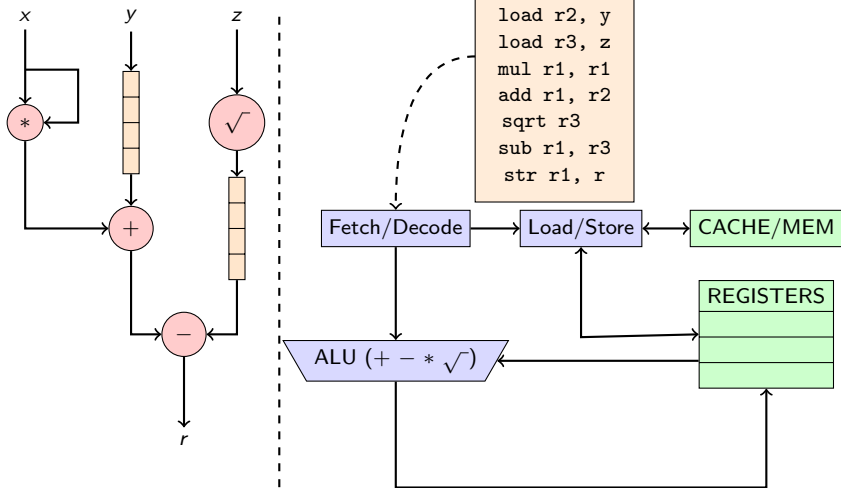
## Kernel application and double dereferencing

```
res_calc(  
    &x[2*edge[2*i]] ,  
    &x[2*edge[2*i+1]] ,  
    &q[4*ecell[2*i]] ,  
    &q[4*ecell[2*i+1]] ,  
    &adt[ecell[2*i]] ,  
    &adt[ecell[2*i+1]] ,  
    &res[4*ecell[2*i]] ,  
    &res[4*ecell[2*i+1]]  
);
```

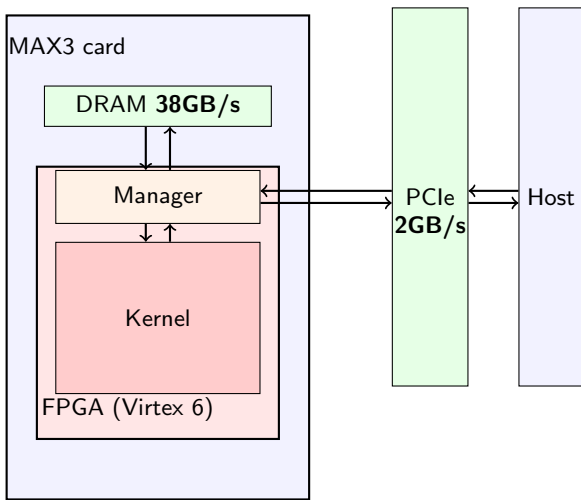


## Why custom streaming?

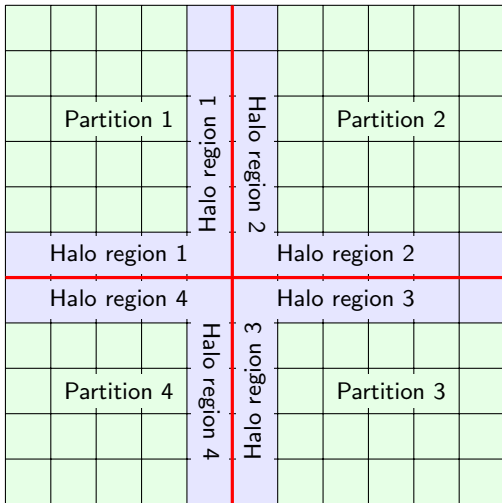
$$r = x^2 + y - \sqrt{z}$$



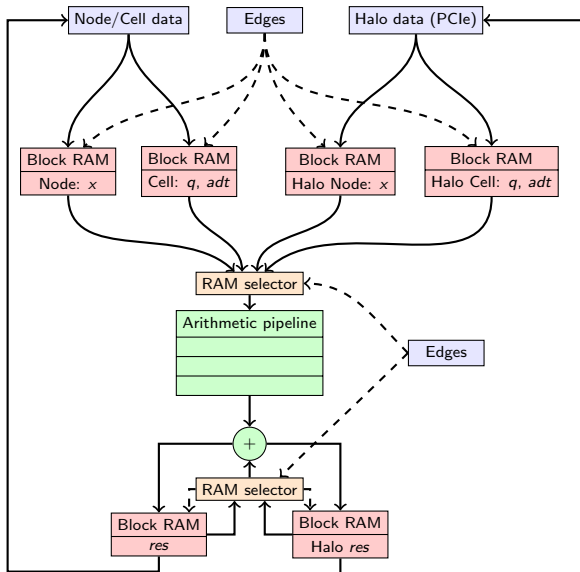
# The hardware



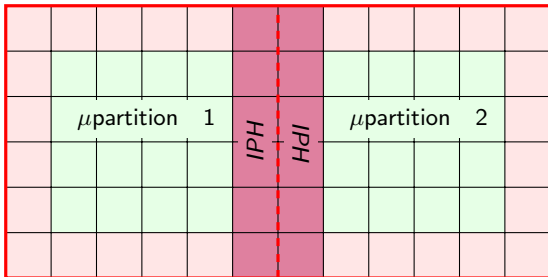
## Partitioning and halos



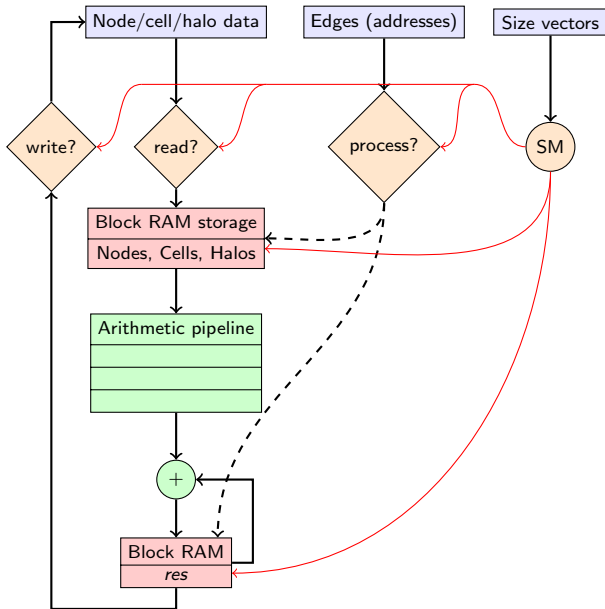
# Architecture design: Accumulation and Halo Exchange



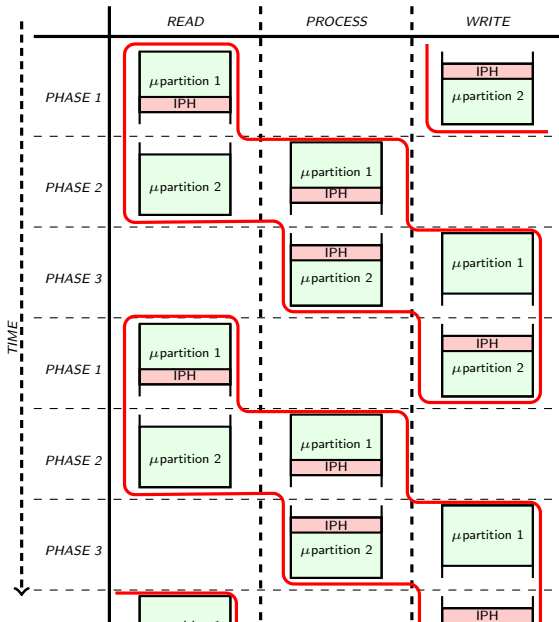
## Two-level partitioning: edge processing and I/O interleaving



## Architecture design: State machine



# Accelerator phases and execution pattern



## Performance Model

We can calculate:

- Time to stream micro-partition from DRAM:

$$t_{DRAM} = \frac{\text{Nonhalo node and cell data}}{\text{DRAM bandwidth}}$$

- Time to stream halo data for micro-partition from PCIe:

$$t_{PCIe} = \frac{\text{Halo node and cell data}}{\text{PCIe bandwidth}}$$

- Time to consume edge data during processing:

$$t_{FPGA} = \frac{\text{Number of edges}}{\text{frequency} \times \text{number of arithmetic pipelines}}$$

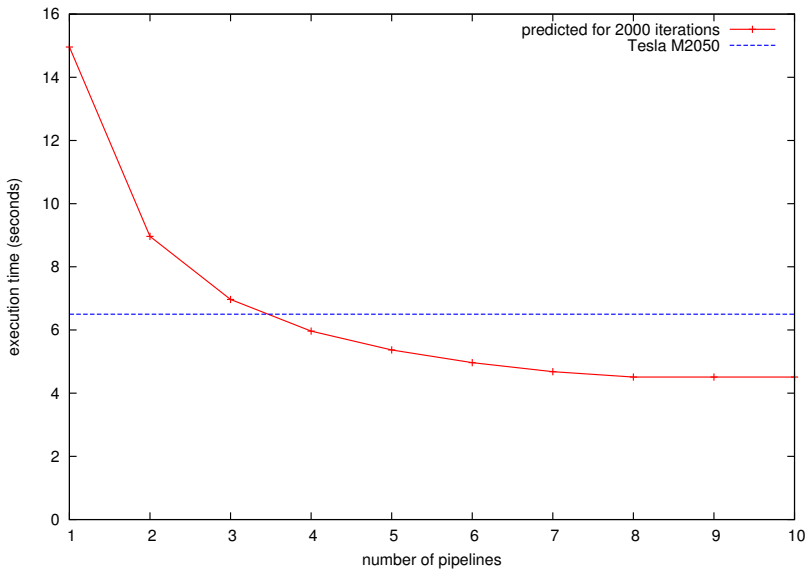
Total time for each phase:  $\max(t_{DRAM}, t_{PCIe}, t_{FPGA})$



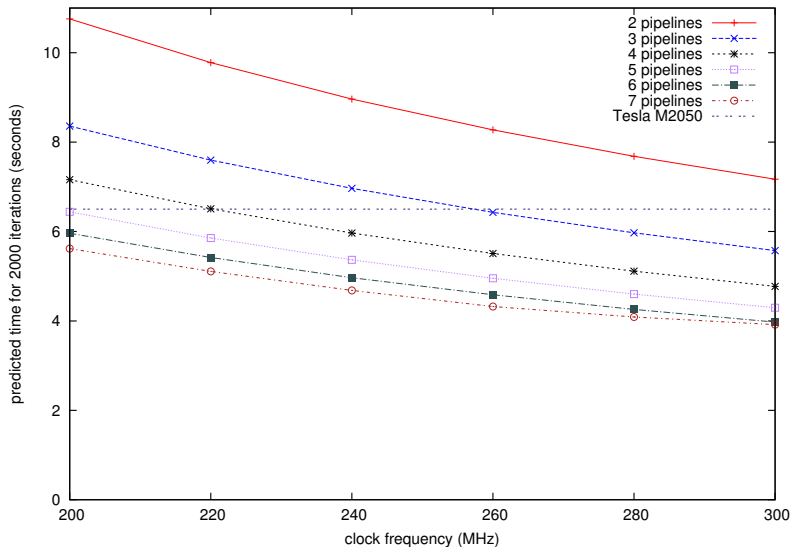
# Design space exploration

- We defined a **family** of architectures.
- We can explore the design space using the performance model to find interesting ones.
- We can vary the problem and architecture parameters and predict the effect on performance.

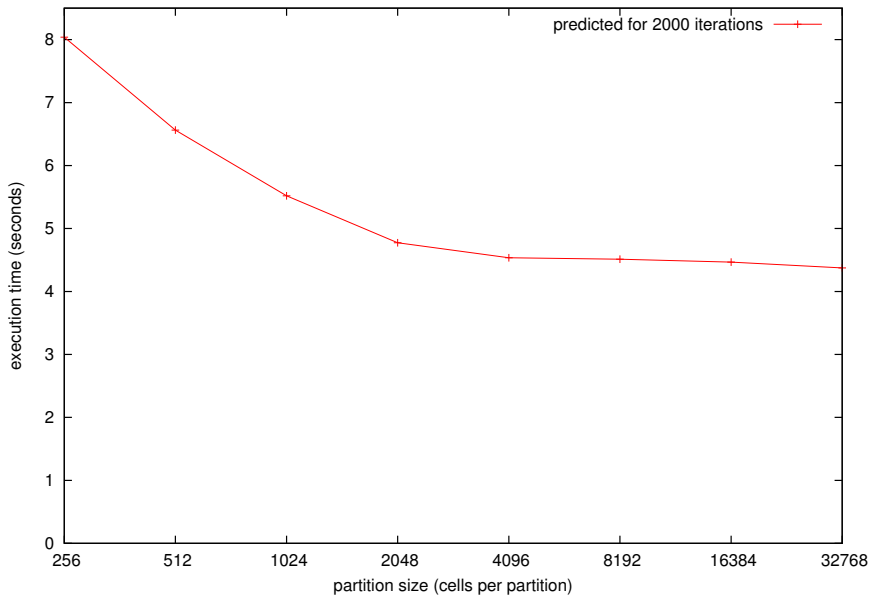
## Execution time vs Number of pipelines



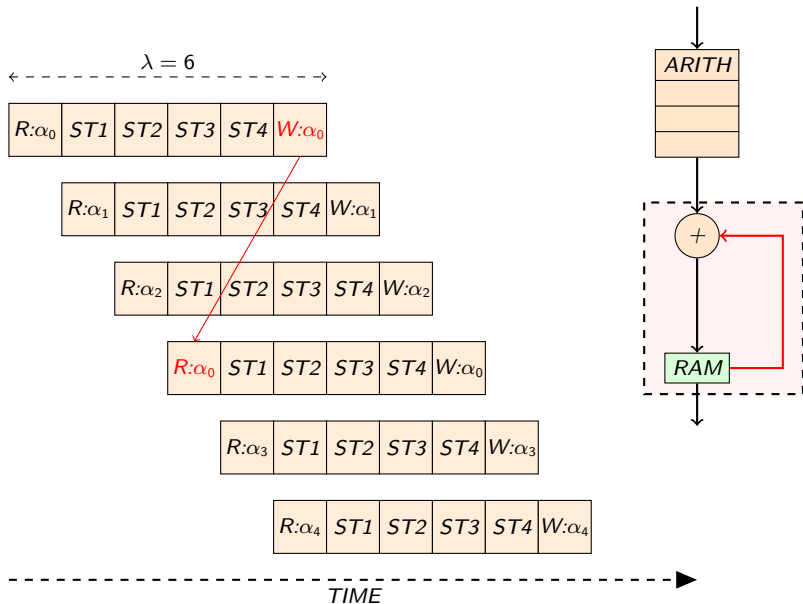
# Execution time vs Number of pipelines and clock frequency



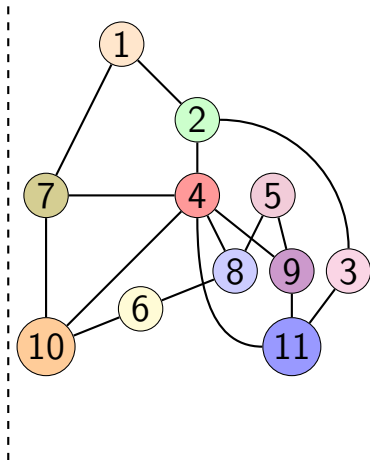
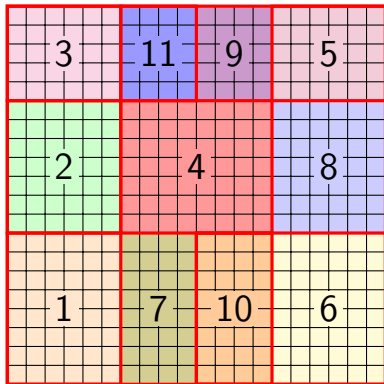
## Execution time vs Partition size



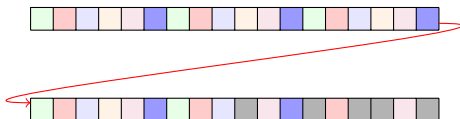
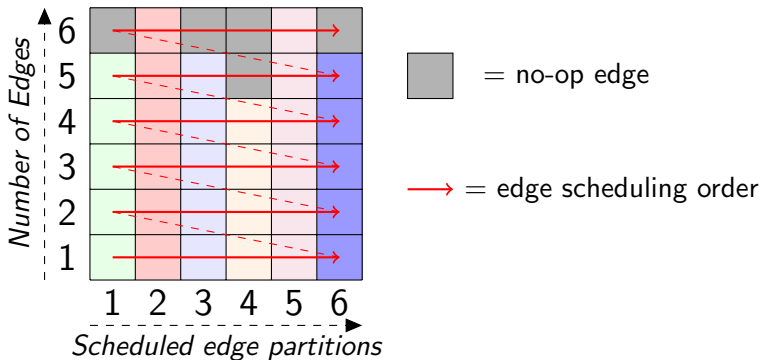
# Implementation issues: Edge dependencies in the pipeline



## Edge-partitions and adjacency graph scheduling



## No-op edges



## Complexity of edge scheduling

- Graph scheduling problem can be expressed as Hamiltonian path problem with extra adjacency constraint.
- **NP-hard!**
- Best we can do is search through the schedule space.
- $O(n!)$  ( $n$  number of nodes in the adjacency graph).



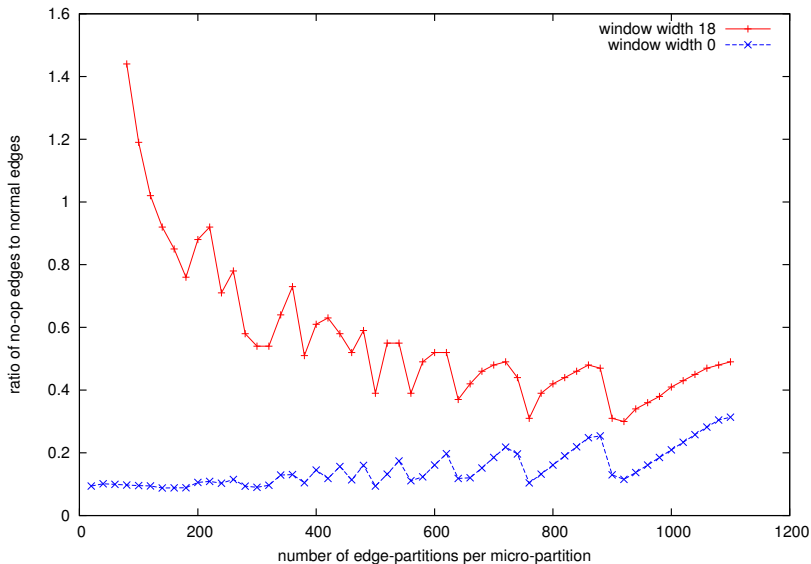
## Graph colouring and no-op edge-partitions

- Can group together partitions with same colour.
- To produce schedule with window-width  $\lambda$  add  $\lambda$  no-op edge-partitions after each colour group. Add  $\lambda \times c$  no-op partitions ( $c$ -number of colours used to colour graph).
- Optimal colouring still **NP**-complete, but we can efficiently find sub-optimal but adequate colouring.
- Greedy graph colouring algorithm. Assign lowest colour not assigned to neighbours of node. Worst-case time complexity is  $O(n^3)$ .

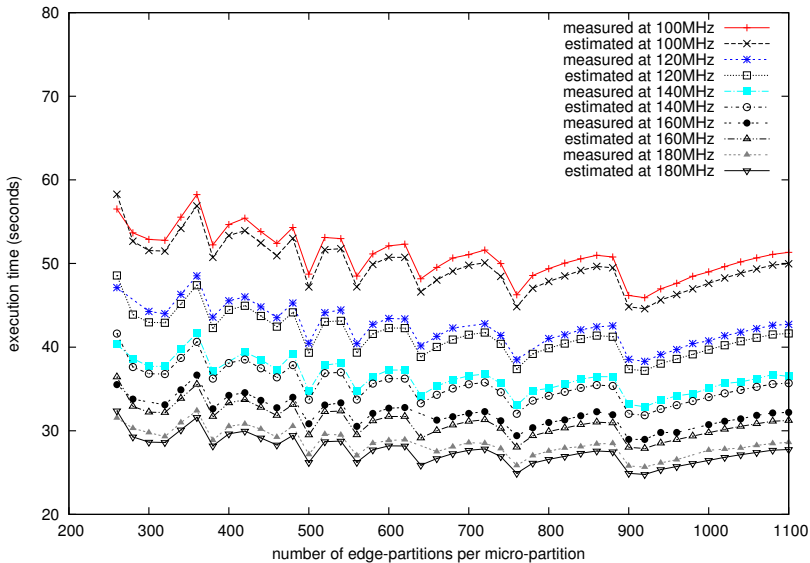
## A note on correctness

- Sample implementation gives wrong arithmetic results.
- Through simulations and debugging tracked down to result committing part of accelerator design.
- *NaNs* from no-op edges committed to result RAMs.
- Kernel consumes and produces correct amount of data in the correct order. Processes correct number of edges.
- *Can still trust the performance results.*

## Evaluation: No-op edges and METIS



## Evaluation: Performance model validation, various frequencies



# Conclusions

- Performance model is validated!
- We can accurately predict the performance of a design space of architectures.
- Simple memory hierarchy provides high predictability. No cache-misses, no non-deterministic thread scheduling by OS.
- Unstructured memory accesses transformed into highly predictable and easily modelled streaming model!
- We showed that an interesting speedup can be achieved.
- Performance rivaling 448-core GPU implementation with only 4-5 pipelines running at a fraction of the clock frequency!

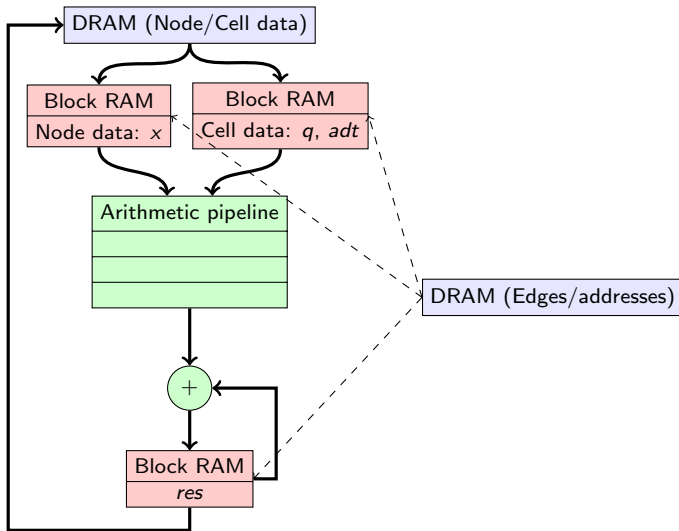
## Further work

- Accelerate other kernels: different element iteration requires different data layout!
- Compilation system: plug in architecture parameters and generate host code and accelerator.
- Data formatting: reduce padding, increase bandwidth utilisation.
- Build multi-pipe designs: model predicts they offer the most performance benefits.
- And more!

Thank you!

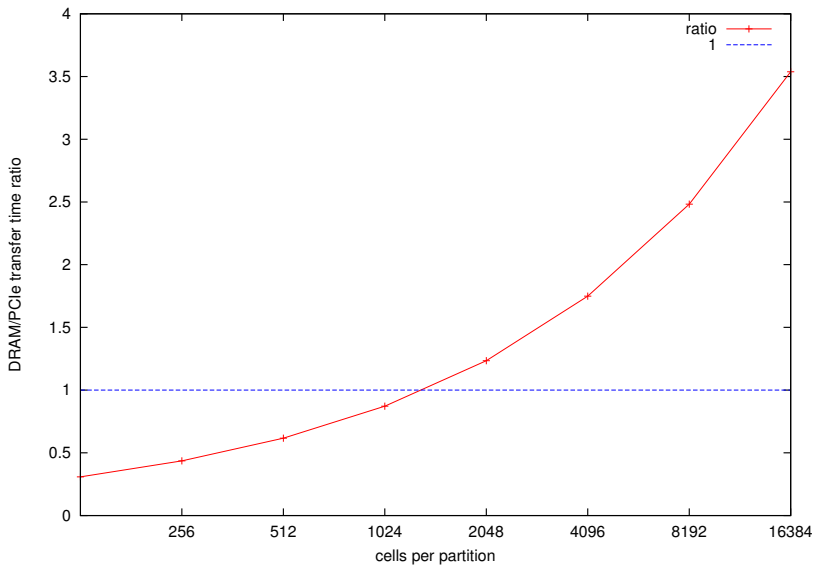
Questions

## Architecture design: 1st iteration

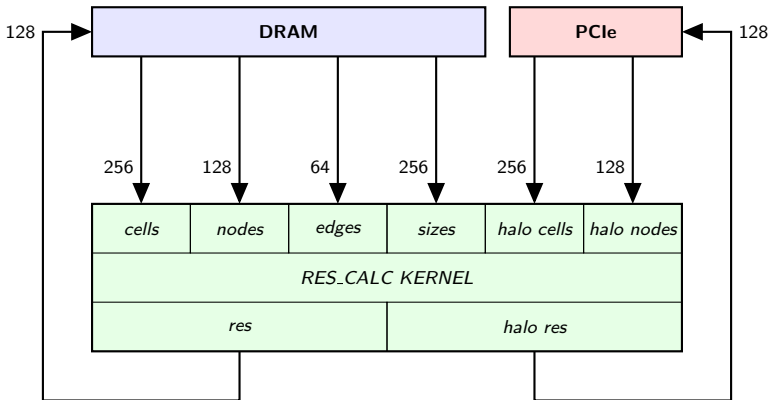




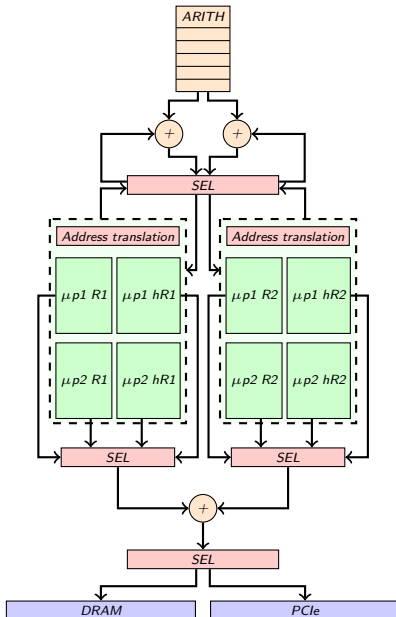
## DRAM/PCIe transfer ratio vs Partition size



## Implementation issues: FPGA accelerator, manager configuration



## Two-port limitation on RAMs



## Host-side halo reduction

